Selected Operations and Applications of n-Tape Weighted Finite-State Machines ⋆

André Kempe

Cadège Technologies
18 rue de Vouillé – 75015 Paris – France
http://a.kempe.free.fr – a.kempe@free.fr

Abstract. A weighted finite-state machine with n tapes (n-WFSM) defines a rational relation on n strings. The paper recalls important operations on these relations, and an algorithm for their auto-intersection. Through a series of practical applications, it investigates the augmented descriptive power of n-WFSMs, w.r.t. classical 1- and 2-WFSMs (acceptors and transducers). Some applications are not feasible with the latter.

1 Introduction

A weighted finite-state machine with n tapes (n-WFSM) [33,7,14,10,12] defines a rational relation on n strings. It is a generalization of weighted acceptors (one tape) and transducers (two tapes).

This paper investigates the potential of n-ary rational relations (resp. n-WFSMs) compared to languages and binary relations (resp. acceptors and transducers), in practical tasks. All described operations and applications have been implemented with Xerox's WFSC tool [17].

The paper is organized as follows: Section 2 recalls some basic definitions about n-ary weighted rational relations and n-WFSMs. Section 3 summarizes some central operations on these relations and machines, such as join and auto-intersection. Unfortunately, due to Post's Correspondence Problem, there cannot exist a fully general auto-intersection algorithm. Section 4 recalls a restricted algorithm for a class of n-WFSMs. Section 5 demonstrates the augmented descriptive power of n-WFSMs through a series of practical applications, namely the morphological analysis of Semitic languages (5.1), the preservation of intermediate results in transducer cascades (5.2), the induction of morphological rules from corpora (5.3), the alignment of lexicon entries (5.4), the automatic extraction of acronyms and their meaning from corpora (5.5), and the search for cognates in a bilingual lexicon (5.6).

⋆ Sections 2–4 are based on published results [18,19,20,4], obtained at Xerox Research Centre Europe (XRCE), Meylan, France, through joint work between Jean-Marc Champarnaud (Rouen Univ.), Jason Eisner (Johns Hopkins Univ.), Franck Guingne and Florent Nicart (XRCE and Rouen Univ.), and the author.

2 Definitions

We recall some definitions about n-ary weighted relations and their machines, following the usual definitions for multi-tape automata [7,6], with semiring weights added just as for acceptors and transducers [24,27]. For more details see [18].

A *weighted n-ary relation* is a function from $(\Sigma^*)^n$ to $\mathbb{K}$, for a given finite alphabet Σ and a given weight semiring $\mathcal{K} = \langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$. A relation assigns a weight to any n-tuple of strings. A weight of $\bar{0}$ can be interpreted as meaning that the tuple is not in the relation. We are especially interested in *rational* (or *regular*) n-ary relations, i.e. relations that can be encoded by n-tape weighted finite-state machines, that we now define.

We adopt the convention that variable names referring to n-tuples of strings include a superscript $^{(n)}$. Thus we write $s^{(n)}$ rather than $\vec{s}$ for a tuple of strings $\langle s_1, \ldots s_n \rangle$. We also use this convention for the names of objects that contain n-tuples of strings, such as n-tape machines and their transitions and paths.

An *n-tape weighted finite-state machine* (n-WFSM) $A^{(n)}$ is defined by a six-tuple $A^{(n)} = \langle \Sigma, Q, \mathcal{K}, E^{(n)}, \lambda, \varrho \rangle$, with Σ being a finite alphabet, Q a finite set of states, $\mathcal{K} = \langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$ the semiring of weights, $E^{(n)} \subseteq (Q \times (\Sigma^*)^n \times \mathbb{K} \times Q)$ a finite set of weighted n-tape transitions, $\lambda : Q \rightarrow \mathbb{K}$ a function that assigns initial weights to states, and $\varrho : Q \rightarrow \mathbb{K}$ a function that assigns final weights to states.

Any transition $e^{(n)} \in E^{(n)}$ has the form $e^{(n)} = \langle y, \ell^{(n)}, w, t \rangle$. We refer to these four components as the transition's source state $y(e^{(n)}) \in Q$, its label $\ell(e^{(n)}) \in (\Sigma^*)^n$, its weight $w(e^{(n)}) \in \mathbb{K}$, and its target state $t(e^{(n)}) \in Q$. We refer by $E(q)$ to the set of out-going transitions of a state $q \in Q$ (with $E(q) \subseteq E^{(n)}$).

A *path* $\gamma^{(n)}$ of length $k \geq 0$ is a sequence of transitions $e_1^{(n)} e_2^{(n)} \cdots e_k^{(n)}$ such that $t(e_i^{(n)}) = y(e_{i+1}^{(n)})$ for all $i \in [1, k-1]$. The label of a path is the element-wise concatenation of the labels of its transitions. The weight of a path $\gamma^{(n)}$ is

$$w(\gamma^{(n)}) \ =_{\text{def}} \ \lambda(y(e_1^{(n)})) \otimes \left(\bigotimes_{j \in [1,k]} w\left(e_j^{(n)}\right) \right) \otimes \varrho(t(e_k^{(n)})) \tag{1}$$

The path is said to be *successful*, and to *accept* its label, if $w(\gamma^{(n)}) \neq \bar{0}$.

3 Operations

We now recall some central operations on n-ary weighted relations and n-WFSMs [21]. The auto-intersection operation was introduced, with the aim of simplifying the computation of the join operation. The notation is inspired by relational databases. For mathematical details of simple operations see [18].

3.1 Simple Operations

Any n-ary weighted rational relation can be constructed by combining the basic rational operations of *union*, *concatenation* and *closure*. Rational operations can

be implemented by simple constructions on the corresponding non-deterministic n-tape WFSMs [34]. These n-tape constructions and their semiring-weighted versions are exactly the same as for acceptors and transducers, since they are indifferent to the n-tuple transition labels.

The *projection* operator $\pi_{\langle j_1,\ldots j_m\rangle}$, with $j_1,\ldots j_m \in [1,n]$, maps an n-ary relation to an m-ary one by retaining in each tuple components specified by the indices $j_1,\ldots j_m$ and placing them in the specified order. Indices may occur in any order, possibly with repeats. Thus the tapes can be permuted or duplicated: $\pi_{\langle 2,1\rangle}$ inverts a 2-ary relation. The *complementary projection* operator $\overline{\pi}_{\{j_1,\ldots j_m\}}$ removes the tapes $j_1,\ldots j_m$ and preserves the order of other tapes.

3.2 Join operation

The n-WFSM *join* operator differs from database join in that database columns are named, whereas our tapes are numbered. Since tapes must explicitly be selected by number, join is neither associative nor commutative.

For any distinct $i_1,\ldots i_r \in [1,n]$ and any distinct $j_1,\ldots j_r \in [1,m]$, we define a *join* operator $\bowtie_{\{i_1=j_1,\ldots i_r=j_r\}}$. It combines an n-ary and an m-ary relation into an $(n+m-r)$-ary relation defined as follows:[1]

$$\left(\mathcal{R}_1^{(n)} \bowtie_{\{i_1=j_1,\ldots i_r=j_r\}} \mathcal{R}_2^{(m)}\right)(\langle u_1,\ldots u_n, s_1,\ldots s_{m-r}\rangle) \;=_{\mathrm{def}}\; \mathcal{R}_1^{(n)}(u^{(n)}) \otimes \mathcal{R}_2^{(m)}(v^{(m)}) \qquad (2)$$

$v^{(m)}$ being the unique tuple s. t. $\overline{\pi}_{\{j_1,\ldots j_r\}}(v^{(m)}) = s^{(m-r)}$ and $(\forall k \in [1,r])\, v_{j_k} = u_{i_k}$.

Important special cases of join are crossproduct $\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)} = \mathcal{R}_1^{(n)} \bowtie_{\emptyset} \mathcal{R}_2^{(m)}$, intersection $\mathcal{R}_1^{(n)} \cap \mathcal{R}_2^{(n)} = \mathcal{R}_1^{(n)} \bowtie_{\{1=1,\ldots n=n\}} \mathcal{R}_2^{(n)}$, and transducer composition $\mathcal{R}_1^{(2)} \circ \mathcal{R}_2^{(2)} = \overline{\pi}_{\{2\}}(\mathcal{R}_1^{(2)} \bowtie_{\{2=1\}} \mathcal{R}_2^{(2)})$.

Unfortunately, rational relations are *not* closed under arbitrary joins [18]. Since the join operation is very useful in practical applications (Sec. 5), it is helpful to have even a partial algorithm: hence our motivation for studying auto-intersection.

3.3 Auto-Intersection

For any distinct $i_1, j_1, \ldots\ i_r, j_r \in [1,n]$, we define an *auto-intersection* operator $\sigma_{\{i_1=j_1, i_2=j_2, \ldots i_r=j_r\}}$. It maps a relation $\mathcal{R}^{(n)}$ to a subset of that relation, preserving tuples $s^{(n)}$ whose elements are equal in pairs as specified, but removing other tuples from the support of the relation.[2] The formal definition is:

$$\left(\sigma_{\{i_1=j_1,\ldots i_r=j_r\}}(\mathcal{R}^{(n)})\right)(\langle s_1,\ldots s_n\rangle) \;=_{\mathrm{def}}\; \begin{cases} \mathcal{R}^{(n)}(\langle s_1,\ldots s_n\rangle) & \text{if } (\forall k \in [1,r]) s_{i_k} = s_{j_k} \\ \bar{0} & \text{otherwise} \end{cases} \qquad (3)$$

[1] For example the tuples $\langle abc, def, \epsilon\rangle$ and $\langle def, ghi, \epsilon, jkl\rangle$ combine in the join $\bowtie_{\{2=1,3=3\}}$ and yield the tuple $\langle abc, def, \epsilon, ghi, jkl\rangle$, with a weight equal to the product of their weights.

[2] The requirement that the $2r$ indices be distinct mirrors the similar requirement on join and is needed in (5). But it can be evaded by duplicating tapes: the illegal operation $\sigma_{\{1=2,2=3\}}(\mathcal{R})$ can be computed as $\overline{\pi}_{\{3\}}(\sigma_{\{1=2,3=4\}}(\pi_{\langle 1,2,2,3\rangle}(\mathcal{R})))$.

It is easy to check that auto-intersecting a relation is different from joining the relation with its own projections. Actually, join and auto-intersection are related by the following equalities:

$$\mathcal{R}_1^{(n)} \bowtie_{\{i_1=j_1,\dots i_r=j_r\}} \mathcal{R}_2^{(m)} \;=\; \overline{\pi}_{\{n+j_1,\dots n+j_r\}} \Big(\sigma_{\{i_1=n+j_1,\dots i_r=n+j_r\}}(\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)}) \Big) \tag{4}$$

$$\sigma_{\{i_1=j_1,\dots i_r=j_r\}}(\mathcal{R}^{(n)}) \;=\; \mathcal{R}^{(n)} \bowtie_{\{i_1=1,j_1=2,\dots i_r=2r-1,j_r=2r\}} \left(\underbrace{(\pi_{\langle 1,1\rangle}(\Sigma^*) \times \cdots \times \pi_{\langle 1,1\rangle}(\Sigma^*)}_{r \text{ times}} \right) \tag{5}$$

Thus, for any class of difficult join instances whose results are non-rational or have undecidable properties [18], there is a corresponding class of difficult auto-intersection instances, and vice-versa. Conversely, a partial solution to one problem would yield a partial solution to the other.

An auto-intersection on a single pair of tapes is said to be a *single-pair* one. An auto-intersection on multiple pairs of tapes can be defined in terms of multiple single-pair auto-intersections:

$$\sigma_{\{i_1=j_1,\dots i_r=j_r\}}(\,\mathcal{R}^{(n)}\,) \;\;=_{\text{def}}\;\; \sigma_{\{i_r=j_r\}}(\,\cdots \sigma_{\{i_1=j_1\}}(\,\mathcal{R}^{(n)}\,)\cdots\,) \tag{6}$$

4 Compilation of Auto-Intersection

We now briefly recall a single-pair auto-intersection algorithm and the class of bounded delay auto-intersections that this algorithm can handle. For a detailed exposure see [19].

4.1 Post's Correspondence Problem

Unfortunately, auto-intersection (and hence join) can be reduced to Post's Correspondence Problem (PCP) [31]. Actually, any PCP instance can be represented as an unweighted 2-FSM, and the set of all solutions to the instance equals the auto-intersection of the 2-FSM [18].

Since it can generally not be decided whether any solution exists to an arbitrary PCP instance, it is also undecidable whether the result of auto-intersection is empty. Therefore, no partial auto-intersection algorithm can be "complete" in the sense that it always returns a correct n-FSM if it is rational, and always terminates with an error code otherwise. Such an algorithm would make PCP generally decidable since a returned n-FSM can always be tested for emptiness, and an error code indicates non-rationality and hence non-emptiness.

4.2 A class of rational auto-intersections

Although there cannot exist a fully general algorithm, $A^{(n)} = \sigma_{\{i=j\}}(A_1^{(n)})$ can be compiled for a class of triples $\langle A_1^{(n)}, i, j\rangle$ whose definition is based on the notion of *delay* [8,26]. The delay $\delta_{\langle i,j\rangle}(s^{(n)})$ is the difference of length of the strings s_i

and s_j of the tuple $s^{(n)}$: $\delta_{\langle i,j\rangle}(s^{(n)}) = |s_i| - |s_j|$ $(i, j \in [1, n])$. We call the delay *bounded* if its absolute value does not exceed some limit. The delay of a path $\gamma^{(n)}$ results from its labels on tapes i and j: $\delta_{\langle i,j\rangle}(\gamma^{(n)}) = |(\ell(\gamma^{(n)}))_i| - |(\ell(\gamma^{(n)}))_j|$. A path has bounded delay if all its prefixes have bounded delay,[3] and an n-WFSM has bounded delay if all its successful paths have bounded delay.

As earlier reported [19], if an n-WFSM $A_1^{(n)}$ does not contain a path traversing both a cycle with positive and a cycle with negative delay w.r.t. tapes i and j,[4] then the delay of all paths of its auto-intersection $A^{(n)} = \sigma_{\{i=j\}}(A_1^{(n)})$ is bounded by some $\delta^{\max}_{\langle i,j\rangle}$, and this bound can be compiled from $A_1^{(n)}$.

4.3 An auto-intersection algorithm

Our algorithm for the above mentioned class of rational auto-intersections proceeds in three steps [19,20] :

1. Test whether the triple $\langle A_1^{(n)}, i, j\rangle$ fulfills the above conditions.
   If not, then the algorithm exits with an error code.
2. Calculation of the bound $\delta^{\max}_{\langle i,j\rangle}$ for the delay of the auto-intersection

   $A^{(n)} = \sigma_{\{i=j\}}(A_1^{(n)})$.
3. Construction of the auto-intersection within the bound.

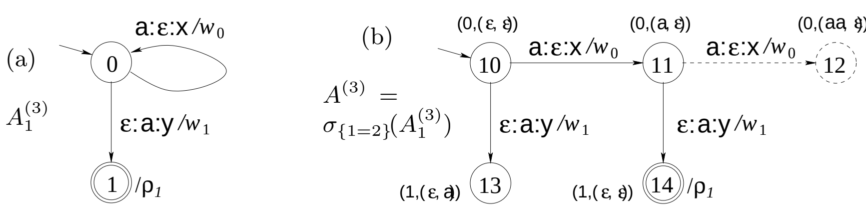

Fig. 1. (a) A 3-WFSM and (b) its auto-intersection

Figure 1 illustrates step 3 of the algorithm: State 0, the initial state of $A_1^{(3)}$, is copied as initial state 10 to $A^{(3)}$. Its annotation, $\langle 0, \langle \varepsilon, \varepsilon\rangle\rangle$, indicates that it is a copy of state 0 and has leftover strings $\langle \varepsilon, \varepsilon\rangle$. Then, all out-going transitions of state 0 and their target states are copied to $A^{(3)}$, as states 11 and 13. A transitions is copied with its original label and weight. The annotation of state 11

[3] Any finite path has bounded delay (since its label is of finite length). An infinite path (traversing cycles) may have bounded or unbounded delay. For example, the delay of a path labeled with $(\langle ab, \varepsilon\rangle\langle \varepsilon, xz\rangle)^h$ is bounded by 2 for any h, whereas that of a path labeled with $\langle ab, \varepsilon\rangle^h \langle \varepsilon, xz\rangle^h$ is unbounded for $h \longrightarrow \infty$.

[4] Note that the n-WFSM may have cycles of both types, but not on the same path.

indicates that it is a copy of state 0 and has leftover strings $\langle \mathsf{a}, \varepsilon \rangle$. These leftover strings result from concatenating the leftover strings of state 10, $\langle \varepsilon, \varepsilon \rangle$, with the relevant components, $\langle \mathsf{a}, \varepsilon \rangle$, of the transition label $\mathsf{a}{:}\varepsilon{:}\mathsf{x}$. For each newly created state $q \in Q_A$, we access the corresponding state $q_1 \in Q_{A_1}$, and copy q_1's outgoing transitions with their target states to $A^{(3)}$, until all states of $A^{(3)}$ have been processed.

State 12 is not created because the delay of its leftover strings $\langle \mathsf{aa}, \varepsilon \rangle$ exceeds the pre-calculated bound of $\delta^{\mathsf{max}}_{\langle 1,2 \rangle} = 1$. The longest common prefix of the two leftover strings of a state is removed. Hence state 14 has leftover strings $\langle \varepsilon, \varepsilon \rangle$ instead of $\langle \mathsf{a}, \varepsilon \rangle \langle \varepsilon, \mathsf{a} \rangle = \langle \mathsf{a}, \mathsf{a} \rangle$. A final state is copied with its original weight if it has leftover strings $\langle \varepsilon, \varepsilon \rangle$, and with weight $\bar{0}$ otherwise. Therefore, state 14 is final and state 13 is not.

The construction is proven to be correct and to terminate [19,20]. It can be performed simultaneously on multiple pairs of tapes.

5 Applications

This section focuses on demonstrating the augmented descriptive power n-WFSMs, w.r.t. to 1- and 2-WFSMs (acceptors and transducers), and on exposing the practical importance of the join operation. It also aims at illustrating how to use n-WFSMs, in practice. Indeed, some of the applications are not feasible with 1- and 2-WFSMs. The section does not focus on the presented applications *per se*.

5.1 Morphological Analysis of Semitic Languages

n-WFSMs have been used in the morphological analysis of Semitic languages [14,22,23, e.g.].

Table 1 by Kiraz [22] shows the "synchronization" of the quadruple $s^{(4)} = \langle \mathsf{aa}, \mathsf{ktb}, \mathsf{waCVCVC}, \mathsf{wakatab} \rangle$ in a 4-WFSM representing an Arabic morphological lexicon. Its first tape encodes a word's vowels, its second the consonants (representing the root), its third the affixes and the templatic pattern (defining how to combine consonants and vowels), and its fourth the word's surface form.

Any of the tapes can be used for input or output. For example, for a given root and vowel sequence, we can obtain all existing surface forms and templates. For a given root and template, we can obtain all existing vowel sequences and surface forms, etc.

			a		a		vocalism
		k		t		b	root
w	a	C	V	C	V	C	pattern and affixes
w	a	k	a	t	a	b	surface form

Table 1. Multi-tape-based morphological anaysis of Arabic; table adapted from Kiraz [22]

5.2 Intermediate Results in Transduction Cascades

Transduction cascades have been extensively used in language and speech processing [1,29,25, e.g.].

In a classical weighted transduction cascade (Figure 2), consisting of transducers $T_1^{(2)} \dots\ T_r^{(2)}$, a weighted input language $L_0^{(1)}$, consisting of one or more words, is composed with the first transducer, $T_1^{(2)}$, on its input tape. The output projection of this composition is the first intermediate result, $L_1^{(1)}$. It is further composed with the second transducer, $T_2^{(2)}$, which leads to the second intermediate result, $L_2^{(1)}$, etc.. Generally, $L_i^{(1)} = \pi_{\langle 2\rangle}(L_{i-1}^{(1)} \diamond T_i^{(2)})$ $(i \in [1, r])$. The output projection of the last transducer is the final result, $L_r^{(1)}$.

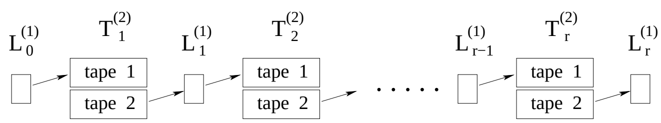

Fig. 2. Classical 2-WFSM transduction cascade

At any point in the cascade, previous intermediate results cannot be accessed. This holds also if the cascade is composed into a single transducer: $T^{(2)} = T_1^{(2)} \diamond \cdots \diamond T_r^{(2)}$. None of the "incorporated" sub-relations of $T^{(2)}$ can refer to a sub-relation other than its immediate predecessor.

In multi-tape transduction cascade, consisting of n-WFSMs $A_1^{(n_1)} \dots A_r^{(n_r)}$, any intermediate results can be preserved and used by subsequent transductions. Figure 3 shows an example where two previous results are preserved at each point, i.e., each intermediate result, $L_i^{(2)}$, has two tapes. The projection of the output tape of the last n-WFSM is the final result, $L_r^{(1)}$:

$$L_1^{(2)} = L_0^{(1)} \bowtie_{\{1=1\}} A_1^{(2)} \tag{7}$$

$$L_i^{(2)} = \pi_{\langle 2,3\rangle}(\ L_{i-1}^{(2)} \bowtie_{\{1=1,2=2\}} A_i^{(3)}\ ) \qquad (i \in [2, r-1]) \tag{8}$$

$$L_r^{(1)} = \pi_{\langle 3\rangle}(\ L_{r-1}^{(2)} \bowtie_{\{1=1,2=2\}} A_r^{(3)}\ ) \tag{9}$$

This augmented descriptive power is also available if the whole cascade is joined into a single 2-WFSM, $A^{(2)}$, although $A^{(2)}$ has only two tapes (in this example), for input and output, respectively. $A^{(2)}$ can be iteratively constructed (Any $B_i^{(m)}$ is the join of $A_1^{(2)}$ to $A_i^{(3)}$) :

$$B_1^{(2)} = A_1^{(2)} \tag{10}$$

$$B_i^{(3)} = \pi_{\langle 1,n-1,n\rangle}(\ B_{i-1}^{(m)} \bowtie_{\{n-1=1,n=2\}} A_i^{(3)}\ ) \qquad (i \in [2, r]\,,\ m \in \{2, 3\}) \tag{11}$$

$$A^{(2)} = \pi_{\langle 1,n\rangle}(\ B_r\ ) \tag{12}$$

Each (except the first) of the "incorporated" multi-tape sub-relations in $A^{(2)}$ will still refer to its two predecessors.

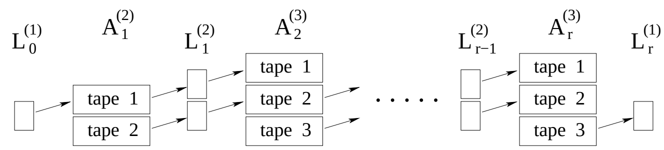

Fig. 3. n-WFSM transduction cascade

5.3 Induction of Morphological Rules

Induction of morphemes and morphological rules from corpora, both supervised and unsupervised, is a subfield of NLP on its own [3,9,5, e.g.]. We do not propose a new method for inducing rules, but rather demonstrate how known steps can be conveniently performed in the framework of n-ary relations.

Learning morphological rules from a raw corpus can include, among others: (1) generating the least costly rule for a given word pair, that rewrites one word to the other, (2) identifying the set of pairs over all corpus words where a given rule applies, and (3) rewriting a given word by means of one or several rules.

Construction of a rule generator For any word pair, such as $\langle parl\mathbf{er}, parl\mathbf{ons}\rangle$ (French, *[to] speak*, *[we] speak*), the generator shall provide a rule, such as "$\mathsf{.er{:}ons}$", suitable for rewriting the first to the second word at minimal cost. In a rule, a dot shall mean that one or more letters remain unmodified, and an $\mathsf{x{:}y}$-part that substring x is replaced by substring y.
We begin with a 4-WFSM that defines rewrite operations:

$$A_1^{(4)} = \Big(\langle\langle ?, ?, ., \mathsf{K}\rangle_{\{1=2\}}, 0\rangle \cup \langle\langle ?, \varepsilon, ?, \mathsf{D}\rangle_{\{1=3\}}, 0\rangle \cup \langle\langle \varepsilon, ?, ?, \mathsf{I}\rangle_{\{2=3\}}, 0\rangle \cup \langle\langle \varepsilon, \varepsilon, :, \mathsf{S}\rangle, 0\rangle\Big)^* \qquad (13)$$

where $?$ can be instantiated by any symbol, ε is the empty string, $_{\{i=j\}}$ a constraint requiring the $?$'s on tapes i and j to be instantiated by the same symbol [28],[5] and 0 a weight over the tropical semiring.

Figure 4 shows the graph of $A_1^{(4)}$ and Figure 5 (rows 1–4) the purpose of its tapes: Tapes 1 and 2 accept any word pair, tape 3 generates a preliminary form of the rule, and tape 4 generates a sequence of preliminary operation codes. The following four cases can occur when $A_1^{(4)}$ reads a word pair (cf. Eq. 13) :

[5] Deviating from [28], we denote symbol constraints similarly to join and auto-intersection constraints.

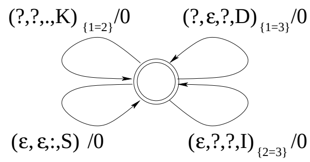

Fig. 4. Initial form $A_1^{(4)}$ of the rule generator

word 1	s w u m
word 2	s w i m
preliminary rule	. . u m : i m
preliminary op. codes	K K D D S I I
final rule	. u m : i m
final operation codes	K k D d S I i
weights	1 0 4 2 0 4 2

Fig. 5. Mapping from the word pair $\langle swum, swim \rangle$ to various sequences

1. $\langle \mathsf{?}, \mathsf{?}, \mathsf{.}, \mathsf{K} \rangle_{\{1=2\}}$: two identical letters are accepted, meaning a letter is kept from word 1 to word 2, which is represented by a "**.**" in the rule and K (keep) in the operation codes,
2. $\langle \mathsf{?}, \varepsilon, \mathsf{?}, \mathsf{D} \rangle_{\{1=3\}}$: a letter is deleted from word 1 to 2, expressed by this letter in the rule and D (delete) in the operation codes,
3. $\langle \varepsilon, \mathsf{?}, \mathsf{?}, \mathsf{I} \rangle_{\{2=3\}}$: a letter is inserted from word 1 to 2, expressed by this letter in the rule and I (insert) in the operation codes
4. $\langle \varepsilon, \varepsilon, \mathsf{:}, \mathsf{S} \rangle$: no letter is matched in either word, a "**:**" is inserted in the rule, and a S (separator) in the operation codes.

Next, we compile $C_1^{(1)}$ that constrains the order of operation codes. For example, D must be followed by S, I must be preceded by S, etc. The constraints are enforced through join (Fig. 5 row 4) : $A_2^{(4)} = A_1^{(4)} \bowtie_{\{4=1\}} C^{(1)}$.

Then, we create $B_1^{(2)}$ that maps temporary rules to their final form by replacing a sequence of dots (longest match) by a single dot. We join $B_1^{(2)}$ with the previous result (Fig. 5 rows 3, 5) : $A_3^{(5)} = A_2^{(4)} \bowtie_{\{3=1\}} B_1^{(2)}$.

Next, we compile $B_2^{(2)}$ that creates more fine-grained operation codes. In a sequence of equal capital letters, it replaces each but the first one with its small form. For example, DDD becomes Ddd. $B_1^{(2)}$ is joined with the previous result (Fig. 5 rows 4, 6) : $A_4^{(6)} = A_3^{(5)} \bowtie_{\{4=1\}} B_2^{(2)}$.

$C_1^{(1)}$, $B_1^{(2)}$, and $B_2^{(2)}$ can be compiled as unweighted automata with a tool such as XFST [13,2] and then be enhanced with neutral weights.

Finally, we assigns weights to the fine-grained operation codes by joining $B_3^{(1)} = (\langle \mathsf{K}, 1 \rangle \cup \langle \mathsf{k}, 0 \rangle \cup \langle \mathsf{D}, 4 \rangle \cup \langle \mathsf{d}, 2 \rangle \cup \langle \mathsf{I}, 4 \rangle \cup \langle \mathsf{i}, 2 \rangle \cup \langle \mathsf{S}, 0 \rangle)^*$ with the previous result (Fig. 5 rows 6, 7) : $A_5^{(6)} = A_4^{(6)} \bowtie_{\{6=1\}} B_3^{(1)}$.

We keep only the tapes of the word pair and of the final rule in the generator (Fig. 5 rows 1, 2, 5). All other tapes are of no further use:

$$G^{(3)} = \pi_{\langle 1,2,5 \rangle} \left(A_5^{(6)} \right) \tag{14}$$

The rule generator $G^{(3)}$ maps any word pair to a finite number of rewrite rules with different weight, expressing the cost of edit operations. The optimal rule (with minimal weight) can be found through n-tape best-path search [16].

Using rewrite rules We suppose that the rules generated from random word pairs undergo some statistical selection process that aims at retaining only meaningful rules.

To facilitate the following operations, a rule's representation can be changed from a string, such as $s^{(1)} =$ "$\mathsf{.er{:}ons}$", to a 2-WFSM $r^{(2)}$ encoding the same relation. This is done by joining the rule with the generator: $r^{(2)} = \pi_{\langle 1,2\rangle}\left(G^{(3)} \bowtie_{\{3=1\}} s^{(1)}\right)$. An $r^{(2)}$ resulting from "$\mathsf{.er{:}ons}$", accepts (on tape 1) only words ending in "er" and changes (on tape 2) their suffix to "ons".

Similarly, a 2-WFSM $R^{(2)}$ that encodes all selected rules can be generated by joining the set of all rules (represented as strings) $S^{(1)}$ with the generator: $R^{(2)} = \pi_{\langle 1,2\rangle}\left(\, G^{(3)} \bowtie_{\{3=1\}} S^{(1)} \,\right)$.

To find all pairs $P^{(2)}$ of words from a corpus where a particular rule applies, we compile the automaton $W^{(1)}$ of all corpus words, and compose it on both tapes of $r^{(2)}$: $P^{(2)} = W^{(1)} \circ r^{(2)} \circ W^{(1)}$. Similarly, identifying all word pairs $P'^{(2)}$ over the whole corpus where any of the rules applies (i.e., the set of "valid" pairs) can be obtained through: $P'^{(2)} = W^{(1)} \circ R^{(2)} \circ W^{(1)}$

Rewriting a word $w^{(1)}$ with a single rule $r^{(2)}$ is done by $w_2^{(1)} = \pi_{\langle 2\rangle}(w_1^{(1)} \circ r^{(2)})$ and $w_1^{(1)} = \pi_{\langle 1\rangle}(r^{(2)} \circ w_2^{(1)})$. Similarly, rewriting a word $w^{(1)}$ with all selected rules is done by $W_2^{(1)} = \pi_{\langle 2\rangle}(w_1^{(1)} \circ R^{(2)})$ and $W_1^{(1)} = \pi_{\langle 1\rangle}(R^{(2)} \circ w_2^{(1)})$.

5.4 String Alignment for Lexicon Construction

Suppose, we want to create a (non-weighted) transducer, $D^{(2)}$, from a list of word pairs $s^{(2)}$ of the form $\langle$*inflected form*, *lemma*$\rangle$, e.g., $\langle \mathsf{swum}, \mathsf{swim}\rangle$, such that each path of the transducer is labeled with one of the pairs. We want to use only transition labels of the form $\langle\sigma,\sigma\rangle$, $\langle\sigma,\varepsilon\rangle$, or $\langle\varepsilon,\sigma\rangle$ $(\forall\sigma\in\Sigma)$, while keeping paths as short as possible. For example, $\langle \mathsf{swum}, \mathsf{swim}\rangle$ should be encoded either by the sequence $\langle \mathsf{s},\mathsf{s}\rangle\langle \mathsf{w},\mathsf{w}\rangle\langle \mathsf{u},\varepsilon\rangle\langle\varepsilon,\mathsf{i}\rangle\langle \mathsf{m},\mathsf{m}\rangle$ or by $\langle \mathsf{s},\mathsf{s}\rangle\langle \mathsf{w},\mathsf{w}\rangle\langle\varepsilon,\mathsf{i}\rangle\langle \mathsf{u},\varepsilon\rangle\langle \mathsf{m},\mathsf{m}\rangle$, rather than by the ill-formed $\langle \mathsf{s},\mathsf{s}\rangle\langle \mathsf{w},\mathsf{w}\rangle\langle \mathsf{u},\mathsf{i}\rangle\langle \mathsf{m},\mathsf{m}\rangle$, or the sub-optimal $\langle \mathsf{s},\varepsilon\rangle\langle \mathsf{w},\varepsilon\rangle\langle \mathsf{u},\varepsilon\rangle\langle \mathsf{m},\varepsilon\rangle$ $\langle\varepsilon,\mathsf{s}\rangle\langle\varepsilon,\mathsf{w}\rangle\langle\varepsilon,\mathsf{i}\rangle\langle\varepsilon,\mathsf{m}\rangle$.

We start with a 5-WFSM over the real tropical semiring [11] :

$$A_1^{(5)} \;=\; \left(\, \langle\langle ?,?,?,?,\mathsf{K}\rangle_{\{1=2=3=4\}},0\rangle \cup \langle\langle\varepsilon,?,@,?,\mathsf{I}\rangle_{\{2=4\}},1\rangle \cup \langle\langle ?,\varepsilon,?,@,\mathsf{D}\rangle_{\{1=3\}},1\rangle \,\right)^{*} \qquad (15)$$

where @ is a special symbol representing ε in an alignment, $_{\{1=2=3=4\}}$ a constraint requiring the $\mathsf{?}$'s on tapes 1 to 4 to be instantiated by the same symbol [28], and 0 and 1 are weights.

Figure 6 shows the graph of $A_1^{(5)}$ and Figure 7 (rows 1–5) the purpose of its tapes: Input word pairs $s^{(2)} = \langle s_1, s_2\rangle$ will be matched on tape 1 and 2, and aligned output word pairs generated from tape 3 and 4. A symbol pair $\langle ?,?\rangle$

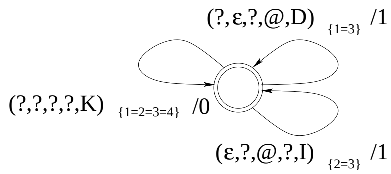

Fig. 6. Initial form $A_1^{(5)}$ of a word pair aligner

input word 1	s	w	u		m
input word 2	s	w		i	m
output word 1	s	w	u	@	m
output word 2	s	w	@	i	m
operation codes	K	K	D	I	K
weights	0	0	1	1	0

Fig. 7. Alignment of the word pair $\langle swum, swim \rangle$

read on tape 1 and 2 is identically mapped to $\langle ?, ? \rangle$ on tape 3 and 4, a $\langle \varepsilon, ? \rangle$ is mapped to $\langle @, ? \rangle$, and a $\langle ?, \varepsilon \rangle$ to $\langle ?, @ \rangle$. $A_1^{(5)}$ will introduce @'s in s_1 (resp. in s_2) at positions where $D^{(2)}$ shall have $\langle \varepsilon, \sigma \rangle$- (resp. a $\langle \sigma, \varepsilon \rangle$-) transitions.[6] Tape 5 generates a sequence of operation codes: K (keep), D (delete), I (insert). For example, $A_1^{(5)}$ will map $\langle \mathsf{swum}, \mathsf{swim} \rangle$, among others, to $\langle \mathsf{swu@m}, \mathsf{sw@im} \rangle$ with KKDIK and to $\langle \mathsf{sw@um}, \mathsf{swi@m} \rangle$ with KKIDK.

To remove redundant (duplicated) alignments, we prohibit an insertion to be immediately followed by a deletion, via the constraint: $C^{(1)} = (\mathsf{K} \cup \mathsf{I} \cup \mathsf{D})^* - (?^* \; \mathsf{I} \; \mathsf{D} \; ?^*)$. The constraint is imposed through join and the operations tape is removed:

$$Aligner^{(4)} = \overline{\pi}_{\{5\}} \Big(\; A_1^{(5)} \bowtie_{\{5=1\}} C^{(1)} \; \Big) \tag{16}$$

The $Aligner^{(4)}$ will map $\langle \mathsf{swum}, \mathsf{swim} \rangle$ among other still to $\langle \mathsf{swu@m}, \mathsf{sw@im} \rangle$ but no to $\langle \mathsf{sw@um}, \mathsf{swi@m} \rangle$. The best alignment (with minimal weight) can be found through n-tape best-path search [16].

5.5 Acronym and Meaning Extraction

The automatic extraction of acronyms and their meaning from corpora is an important sub-task of text mining, and received much attention [37,32,35, e.g.].

It can be seen as a special case of string alignment between a text chunk and an acronym. For example, the chunk "they have many hidden Markov models" can be aligned with the acronym "HMMs" in different ways, such as "they have many <u>h</u>idden <u>M</u>arkov <u>m</u>odel<u>s</u>" or "t<u>h</u>ey have <u>m</u>any hidden <u>M</u>arkov model<u>s</u>". Alternative alignments have different cost, and ideally the least costly one should give the correct meaning.

An alignment-based approach can be implemented by means of a 3-WFSM that reads a text chunk on tape 1 and an acronym on tape 2, and generates all possible alignments on tape 3, inserting dots to mark letters used in the acronym. For the above example this would give "they have many .hidden .Markov .model.s", among others.

[6] Later, we simply replace in $D^{(2)}$ all @ by ε.

The 3-WFSM can be generated from n-ary regular expressions that define the task in as much detail as required (cf. Sec. 5.3 and 5.4). For a detailed description see [15]. The best alignment, i.e., the most likely meaning of an acronym is found through n-tape best-path search [16].

The advantage of aligning via a n-WFSM rather than a classical alignment matrix [36,30] is that the n-WFSM can be built from regular expressions that define very subtle criteria, such as disallowing certain alignments or favoring others based on weights that depend on long-distance context.

5.6 Cognate Search

Extracting cognates with equal meaning from an English-German dictionary $\mathrm{EG}^{(3)}$ that encodes triples $\langle$*English word*, *German word*, *part of speech*$\rangle$, means to identify all paths of $\mathrm{EG}^{(3)}$ that have similar strings on tapes 1 and 2.

We create a similarity automaton $S^{(2)}$ that describes through weights the degree of similarity between English and German words. This can either be expressed through edit distance (cf. Sec. 5.3, 5.4, and 5.5) or through weighted synchronic grapheme correspondences (e.g.: *d*-*t*, *ght*-*cht*, *th*-*d*, *th*-*ss*, …) : $S^{(2)} = \big(\langle\langle ?, ? \rangle_{\{1=2\}}, w_0 \rangle \cup \langle\langle d, t \rangle, w_1 \rangle \cup \langle\langle ght, cht \rangle, w_2 \rangle \cup \ldots \big)^*$
When recognizing an English-German word pair, $S^{(2)}$ accepts either any two equal symbols in the two words (via $\langle ?, ? \rangle_{\{1=2\}}$) or some English sequence and its German correspondence (e.g. *ght* and *cht*) with some weight.

The set of cognates $\mathrm{EG}^{(3)}_{\mathrm{cog}}$ is obtained by joining the dictionary with the similarity automaton: $\mathrm{EG}^{(3)}_{\mathrm{cog}} = \mathrm{EG}^{(3)} \bowtie_{\{1=1,2=2\}} S^{(2)}$
$\mathrm{EG}^{(3)}_{\mathrm{cog}}$ contains all (and only) the cognates with equal meaning in $\mathrm{EG}^{(3)}$ such as $\langle \mathrm{daughter}, \mathrm{tochter}, \mathrm{noun} \rangle$, $\langle \mathrm{eight}, \mathrm{acht}, \mathrm{num} \rangle$, or $\langle \mathrm{light}, \mathrm{leicht}, \mathrm{adj} \rangle$. Weighs of triples express similarity of words.

Note that this result cannot be achieved through ordinary transducer composition. For example, composing $S^{(2)}$ with the English and the German words separately: $\pi_{\langle 1 \rangle}(\mathrm{EG}^{(3)}) \diamond S^{(2)} \diamond \pi_{\langle 2 \rangle}(\mathrm{EG}^{(3)})$, also yields false cognates such as $\langle \mathrm{become}, \mathrm{bekommen} \rangle$ (*[to] obtain*).

6 Conclusion

The paper recalled basic definitions about n-ary weighted relations and their n-WFSMs, central operations on these relations and machines, and an algorithm for the important auto-intersection operation.

It investigated the potential of n-WFSMs, w.r.t. classical 1- and 2-WFSMs (acceptors and transducers), in practical tasks. Through a series of applications, it exposed their augmented descriptive power and the importance of the join operation. Some of the applications are not feasible with 1- or 2-WFSMs.

In the morphological analysis of Semitic languages, n-WFSMs have been used to synchronize the vowels, consonants, and templatic pattern into a surface form. In transduction cascades consisting of n-WFSMs, intermediate result can

be preserved and used by subsequent transductions. n-WFSMs permit not only to map strings to strings or string m-tuples to k-tuples, but m-ary to k-ary string relations, such as an non-aligned word pair to its aligned form, or to a rewrite rule suitable for mapping one word to the other. In string alignment tasks, an n-WFSM provides better control over the alignment process than a classical alignment matrix, since it can be compiled from regular expressions defining very subtle criteria, such as long-distance dependencies for weights.

References

1. S. Aït-Mokhtar and J.-P. Chanod. Incremental finite-state parsing. In *Proc. 5th Int. Conf. ANLP*, pages 72–79, Washington, DC, USA, 1997.
2. K.R. Beesley and L. Karttunen. *Finite State Morphology.* CSLI Publications, Palo Alto, CA, 2003.
3. M. Brent. An efficient, probabilistically sound algorithm for segmentation and word discovery. *Machine Learning*, 34:71–106, 1999.
4. J.-M. Champarnaud, F. Guingne, A. Kempe, and F. Nicart. Algorithms for the join and auto-intersection of multi-tape weighted finite-state machines. *Int. Journal of Foundations of Computer Science*, 19(2):453–476, 2008. World Scientific.
5. M. Creutz and K. Lagus. Unsupervised models for morpheme segmentation and morfology learning. *ACM Transactions on Speech and Language Processing*, 4(1), 2007.
6. S. Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, San Diego, 1974.
7. C.C. Elgot and J.E. Mezei. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965.
8. C. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108(1):45–82, 1993.
9. J. Goldsmith. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27:153–198, 2001.
10. T. Harju and J. Karhumäki. The equivalence problem of multitape finite automata. *Theoretical Computer Science*, 78(2):347–355, 1991.
11. P. Isabelle and A. Kempe. Automatic string alignment for finite-state transducers. *Unpublished work*, 2004.
12. R.M. Kaplan and M. Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.
13. L. Karttunen, T. Gaál, and A. Kempe. *Xerox finite state complier*. Online demo and documentation, 1998. Xerox Research Centre Europe, Grenoble, France. http://www.xrce.xerox.com/competencies/content-analysis/fsCompiler/.
14. M. Kay. Nonconcatenative finite-state morphology. In *Proc. 3rd Int. Conf. EACL*, pages 2–10, Copenhagen, Denmark, 1987.
15. A. Kempe. Acronym-meaning extraction from corpora using multitape weighted finite-state machines. Research report 2006/019, Xerox Research Centre Europe, Meylan, France, 2006.
16. A. Kempe. Viterbi algorithm generalized for n-tape best-path search. In *Proc. 8th Int. Workshop FSMNLP*, Pretoria, South Africa, 2009.
17. A. Kempe, C. Baeijs, T. Gaál, F. Guingne, and F. Nicart. WFSC – A new weighted finite state compiler. In O.H. Ibarra and Z. Dang, editors, *Proc. 8th Int. Conf. CIAA*, volume 2759 of *LNCS*, pages 108–119, Santa Barbara, CA, USA, 2003. Springer Verlag, Berlin, Germany.

18. A. Kempe, J.-M. Champarnaud, and J. Eisner. A note on join and auto-intersection of n-ary rational relations. In B. Watson and L. Cleophas, editors, *Proc. Eindhoven FASTAR Days*, number 04–40 in TU/e CS TR, pages 64–78, Eindhoven, Netherlands, 2004.
19. A. Kempe, J.-M. Champarnaud, J. Eisner, F. Guingne, and F. Nicart. A class of rational n-wfsm auto-intersections. In J. Farré, I. Litovski, and S. Schmitz, editors, *Proc. 10th Int. Conf. CIAA*, pages 266–274, Sophia Antipolis, France, 2005.
20. A. Kempe, J.-M. Champarnaud, F. Guingne, and F. Nicart. Wfsm auto-intersection and join algorithms. In *Proc. 5th Int. Workshop FSMNLP*, Helsinki, Finland, 2005.
21. A. Kempe, F. Guingne, and F. Nicart. Algorithms for weighted multi-tape automata. Research report 2004/031, Xerox Research Centre Europe, Meylan, France, 2004.
22. G.A. Kiraz. Linearization of nonlinear lexical representations. In J. Coleman, editor, *Proc. 3rd ACL SIG Computational Phonology*, Madrid, Spain, 1997.
23. G.A. Kiraz. Multitiered nonlinear morphology using multitape finite automata: a case study on Syriac and Arabic. *Computational Lingistics*, 26(1):77–105, March 2000.
24. W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer Verlag, Berlin, Germany, 1986.
25. S. Kumar and W. Byrne. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proc. Int. Conf. HLT-NAACL*, pages 63–70, Edmonton, Canada, 2003.
26. M. Mohri. Edit-distance of weighted automata. In *Proc. 7th Int. Conf. CIAA*, volume 2608 of *LNCS*, pages 1–23, Tours, France, 2003. Springer Verlag, Berlin, Germany.
27. M. Mohri, F.C.N. Pereira, and M. Riley. A rational design for a weighted finite-state transducer library. *LNCS*, 1436:144–158, 1998.
28. F. Nicart, J.-M. Champarnaud, T. Csáki, T. Gaál, and A. Kempe. Multi-tape automata with symbol classes. In O.H. Ibarra and H.-C. Yen, editors, *Proc. 11th Int. Conf. CIAA*, volume 4094 of *LNCS*, pages 126–136, Taipei, Taiwan, 2006. Springer Verlag.
29. F.C.N. Pereira and M.D. Riley. Speech recognition by composition of weighted finite automata. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*, pages 431–453. MIT Press, Cambridge, MA, USA, 1997.
30. A. Pirkola, J. Toivonen, H. Keskustalo, K. Visala, and K. Järvelin. Fuzzy translation of cross-lingual spelling variants. In *Proc. 26th Annual Int. ACM SIGIR*, pages 345–352, Toronto, Canada, 2003.
31. E. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52:264–268, 1946.
32. J. Pustejovsky, J. Casta no, B. Cochran, M. Kotecki, M. Morrell, and A. Rumshisky. Linguistic knowledge extraction from medline: Automatic construction of an acronym database. In *Proc. 10th World Congress on Health and Medical Informatics (Medinfo 2001)*, 2001.
33. M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.
34. A.L. Rosenberg. On n-tape finite state acceptors. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 76–81, 1964.

35. A. Schwartz and M. Hearst. A simple algorithm for identifying abbreviation definitions in biomedical texts. In *Proc. Pacific Symposium on Biocomputing (PSB-2003)*, 2003.
36. R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, 1974.
37. S. Yeates, D. Bainbridge, and I.H. Witten. Using compression to identify acronyms in text. In *Proc. Data Compression Conf. (DCC-2000)*, Snowbird, Utah, USA, 2000. (Also published in a longer form as Working Paper 00/01, Department of Computer Science, University of Waikato, January 2000).